STUDY OF CORDIC BASED PROCESSING ELEMENT FOR DIGITAL SIGNAL PROCESSING ALGORITHMS

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Technology
In
Electrical Engineering

By
S. SYAM BABU

Department of Electrical Engineering
National Institute of Technology
Rourkela
2007

STUDY OF CORDIC BASED PROCESSING ELEMENT FOR DIGITAL SIGNAL PROCESSING ALGORITHMS

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Technology
In
Electrical Engineering

By

S. Syam Babu

Under the Guidance of
Prof. S. Mohanty

Department of Electrical Engineering
National Institute of Technology
Rourkela
2007

National Institute of Technology
Rourkela

CERTIFICATE

This is to certify that the thesis entitled, "**Study of CORDIC based processing element for digital signal processing algorithms**" submitted by Mr. **S. Syam Babu** in partial fulfillment of the requirements for the award of Master of Technology Degree in **Electrical** Engineering with specialization in "**Electronic System and Communication**" at the National Institute of Technology, Rourkela (Deemed University) is an authentic work carried out by him under my supervision and guidance.

To the best of my knowledge, the matter embodied in the thesis has not been submitted to any other university / institute for the award of any Degree or Diploma.

Date:

Prof. S. Mohanty
Electrical Engineering Department
National Institute of Technology
Rourkela - 769008

ACKNOWLEDGEMENT

I express my sincere gratitude and appreciation to many people who helped keep me on track toward the completion of my thesis. Firstly, I owe the biggest thanks to my supervisor, **Prof. S. Mohanty**, whose advice, patience, and care boosted my morale.

I am very much thankful to our HOD, **Prof. P. K. Nanda**, for providing us with best facilities in the department and his timely suggestions. I also thank all the teaching and non-teaching staff for their cooperation to the students.

My special thanks to Mrs. N. Ramya Bhuvana, who helped me in completion of my thesis. I also thank all my friends, without whose support my life might have been miserable here.

I wish to express my gratitude to my parents, whose love and encouragement have supported me throughout my education.

S. Syam Babu

CONTENTS

ABSTRACT

There is a high demand for the efficient implementation of complex arithmetic operations in many Digital Signal Processing (DSP) algorithms. The COordinate Rotation DIgital Computer (CORDIC) algorithm is suitable to be implemented in DSP algorithms since its calculation for complex arithmetic is simple and elegant. Besides, since it avoids using multiplications, adopting the CORDIC algorithm can reduce the complexity.

Here, in this project CORDIC based processing element for the construction of digital signal processing algorithms is implemented. This is a flexible device that can be used in the implementation of functions such as Singular Value Decomposition (SVD), Discrete Cosine Transform (DCT) as well as many other important functions. It uses a CORDIC module to perform arithmetic operations and the result is a flexible computational processing element (PE) for digital signal processing algorithms. To implement the CORDIC based architectures for functions like SVD and DCT, it is required to decompose their computations in terms of CORDIC operations.

SVD is widely used in digital signal processing applications such as direction estimation, recursive least squares (RLS) filtering and system identification. Two different Jacobi-type methods for SVD parallel computation are usually considered, namely the Kogbetliantz (two-sided rotation) and the Hestenes (one-sided rotation) method. Kogbetliantz's method has been considered, because it is suitable for mapping onto CORDIC array architecture and highly suitable for parallel computation. Here in its implementation, CORDIC algorithm provides the arithmetic units required in the processing elements as these enable the efficient implementation of plane rotation and phase computation. Many fundamental aspects of linear algebra rely on determining the rank of a matrix, making the SVD an important and widely used technique.

DCT is one of the most widely used transform techniques in digital signal processing and it computation involves many multiplications and additions. The DCT based on CORDIC algorithm does not need multipliers. Moreover, it has regularity and simple architecture and it is used to compress a wide variety of images by transferring data into frequency domain. These digital signal-processing algorithms are used in many applications.

The purpose of this thesis is to describe a solution in which a conventional CORDIC system is used to implement an SVD and DCT processing elements. The approach presented combines the low circuit complexity with high performance.

LIST OF FIGURES

Chapter 1

INTRODUCTION

INTRODUCTION

The digital signal processing landscape has long been dominated by microprocessors with enhancements such as multiply-accumulate instructions and special addressing modes. The advent of reconfigurable logic computers permits the higher speeds of dedicated hardware solutions at costs that are competitive with the traditional software approach. Unfortunately, algorithms optimized for these microprocessor-based systems do not usually map well into hardware. While hardware-efficient solutions often exist, the dominance of the software systems has kept those solutions out of the spotlight. Among these hardware-efficient algorithms, there is a class of iterative solutions for trigonometric and other transcendental functions that use only shifts and adds to perform. The trigonometric functions are based on vector rotations and trigonometric algorithm is called CORDIC, an acronym for COordinate Rotation Digital Computer. The trigonometric CORDIC algorithms were originally developed as a digital solution for real time navigation problems.

The CORDIC algorithm can operate in either vectoring or rotation mode. Vectoring mode performs Cartesian to polar conversion. An input vector is rotated until it is on x-axis. The final x value is equal to the magnitude of the input vector. While rotating the vector, the total angle traversed is also recorded, which provides the phase of the input vector. Rotation mode performs polar to Cartesian conversion. The input vector is rotated by the specified angle. The final x and y values in rotation and vectoring modes are scaled by the CORDIC processing gain. This processing gain varies with the number of iterations performed. It approaches to 1.6476 as the number of iterations goes to infinity.

The CORDIC algorithm has found its way into diverse applications including the 8087 math coprocessor and radar signal processors. CORDIC rotation has also been proposed for computing Discrete Fourier, Discrete Cosine, Discrete Hartley and Z-transforms, filtering, Singular Value Decomposition, and solving linear systems.

Singular value decomposition (SVD) is widely used in digital signal processing applications such as direction estimation, spectrum analysis and systems identification. Two different Jacobi-type methods for SVD parallel computation are usually considered, namely the Kogbetliantz (two-sided rotation) and the Hestenes (one-sided rotation) method. The Kogbetliantz's method has been adopted because it is suitable for mapping onto CORDIC array architecture [1] and highly suitable for parallel computation.

In Kogbetliantz's method, the norm of the off-diagonal elements of the matrix is successively reduced by a sequence of two-sided Givens transformations, which requires the computation of a rotation angle and subsequent operations. CORDIC algorithm [2] provides an attractive means for implementing the arithmetic units required in typical SVD processing elements as these enable the efficient implementation of plane rotation and phase computation.

The Discrete Cosine Transform has become one of the most widely used transform techniques in digital signal processing and it is one of the computationally intensive transforms, which require many multiplications and additions. The DCT based on CORDIC [3] algorithm does not need multipliers. Moreover, it has regularity and simple hardware architecture. DCT is an image compression method used in many applications for its effectiveness to compress a wide variety of images by transferring data into frequency domain.

Chapter 2

CORDIC THEORY

2. CORDIC THEORY: AN ALGORITHM FOR VECTOR ROTATION:

All of the trigonometric functions can be computed or derived from functions using vector rotations, as will be discussed in this section. Vector rotation can also be used for polar to rectangular and rectangular to polar conversions, for vector magnitude, and as a building block in certain transforms such as the DFT and DCT. The CORDIC algorithm provides an iterative method of performing vector rotations by arbitrary angles using only shifts and adds. The algorithm credited to Volder [2], is derived from the general (Givens) rotation transform:

$$x' = x\cos\phi - y\sin\phi$$
$$y' = y\cos\phi + x\sin\phi$$

This rotates a vector in a Cartesian plane by the angle ϕ as shown in the figure 2.1

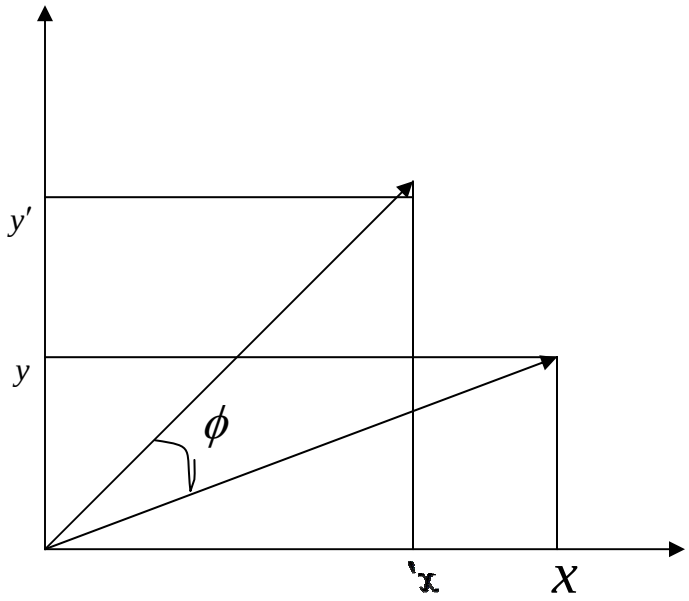

Figure 2.1. Vector rotation

These can be rearranged so that:

$$x' = \cos\phi.[x - y\tan\phi] \qquad (2.1)$$
$$y' = \cos\phi.[y + x\tan\phi]$$

So far, nothing is simplified. However if the rotation angles are restricted such that $\tan(\phi) = \pm 2^{-i}$, the multiplication by the tangent term is reduced to simple shift operation. Arbitrary angles of rotation are obtainable by performing a series of successively smaller elementary rotations. If the decision at each iteration, i, is which direction to rotate rather than whether or not to rotate, then the $\cos(\delta_i)$ term becomes a constant (because $\cos(\delta_i) = \cos(-\delta_i)$). The iterative rotation can now be expressed as:

$$x_{i+1} = k_i[x_i - y_i.d_i.2^{-i}] \qquad (2.2)$$

$$y_{i+1} = k_i[y_i + x_i.d_i.2^{-i}]$$

Where:

$$k_i = \cos(\tan^{-1} 2^{-i}) = 1/\sqrt{1+2^{-2i}}$$

$$d_i = \pm 1$$

Removing the scale constant from the iterative equations yields a shift-add algorithm for vector rotation. The product of the k_i's can be applied elsewhere in the system treated as part of a system processing gain. That product approaches 0.6073 as the number of iterations goes to infinity. Therefore, the rotation algorithm has a gain A_n, of approximately 1.647. The exact gain depends on the number of iterations, and obeys the relation

$$A_n = \prod_n \sqrt{1+2^{-2i}}$$

The angle of a composite rotation is uniquely defined by the sequence of the directions of the elementary rotations. That sequence can be represented by a decision vector. The set of all possible decision vectors is an angular measurement system based on binary arctangents. Conversions between this angular system and any other can be accomplished using a look-up. A better conversion method uses an additional adder-subtractor that accumulates the elementary rotation angles at each iteration. The elementary angles can be expressed in any convenient angular unit. Those angular values are supplied by a small lookup table (one entry per iteration) or are hardwired, depending on the implementation.

The angle accumulator adds a third difference equation to the CORDIC algorithm:

$$z_{i+1} = z_i - d_i . \tan^{-1}(2^{-i}) \qquad (2.3)$$

Obviously, in cases where the angle is useful in the arctangent base, this extra element is not needed.

The CORDIC rotator is normally operated in one of two modes. The first, called rotation by Volder [2] rotates the input vector by a specified angle (given as an argument). The second mode, called vectoring, rotates the input vector to the x-axis while recording the angle required to make that rotation.

In rotation mode, the angle accumulator is initialized with the desired rotation angle. The rotation decision at each iteration is made to diminish the magnitude of the residual angle in the angle accumulator. The decision at each iteration is therefore based on the sign of the residual angle after each step. Naturally, if the input angle is already expressed in the binary arctangent base, the angle accumulator may be eliminated. For rotation mode, the cordic equations are:

$$\begin{aligned} x_{i+1} &= x_i - y_i . d_i . 2^{-i} \\ y_{i+1} &= y_i + x_i . d_i . 2^{-i} \\ z_{i+1} &= z_i - d_i . \tan^{-1}(2^{-i}) \end{aligned} \qquad (2.4)$$

Where d_i =-1 if z_i <0, +1 otherwise

Which provides the following result

$$\begin{aligned} x_n &= A_n[x_0 \cos z_0 - y_0 \sin z_0] \\ y_n &= A_n[y_0 \cos z_0 + x_0 \sin z_0] \\ z_n &= 0 \\ A_n &= \prod_n \sqrt{1 + 2^{-2i}} \end{aligned} \qquad (2.5)$$

In the vectoring mode, the CORDIC rotator rotates the input vector through whatever angle is necessary to align the result vector with the x axis. The result of the vectoring operation is a rotation angle and the scaled magnitude of the original vector (the x component of the result). The vectoring function works by seeking to minimize the

y component of the residual vector at each rotation. The sign of the residual y component is used to determine which direction to rotate next. If the angle accumulator is initialized with zero, it will contain the traversed angle at the end of the iterations. In vectoring mode, the CORDIC equations are:

$$x_{i+1} = x_i - y_i.d_i.2^{-i}$$
$$y_{i+1} = y_i + x_i.d_i.2^{-i}$$
$$z_{i+1} = z_i - d_i.\tan^{-1}(2^{-i})$$

Where d_i =+1 if y_i <0, -1 otherwise.

Then:

$$x_n = A_n\sqrt{x_0^{\,2} + y_0^{\,2}}$$
$$y_n = 0$$
$$z_n = z_0 + \tan^{-1}(y_0 / x_0) \qquad (2.6)$$
$$A_n = \prod_n \sqrt{1+2^{-2i}}$$

The CORDIC rotation and vectoring algorithms as stated are limited to rotation angles between $-\pi/2$ and $\pi/2$. This limitation is due to the use of 2^0 for the tangent in the first iteration. For composite rotation angles larger than $\pi/2$, an additional rotation is required. Volder describes an initial rotation $\pm\pi/2$. This gives the correction iteration:

x'=-d.y
y'=d.x
z'=z + d. $\pi/2$

Where d=+1 if x<0, -1 otherwise.

There is no growth for this initial rotation. Alternatively, an initial rotation of either π or 0 can be made, avoiding the reassignment of the x and y components to the rotator elements. Again, there is no growth due to the initial rotation:

x'=d.x
y'=d.y
z'=z if d=1, or z=-π if d=-1
d=-1 if x<0, +1 otherwise.

The CORDIC rotator described is usable to compute several trigonometric functions directly and others indirectly. Judicious choice of initial values and modes permits direct computation of sine, cosine, arctangent, vector magnitude and transformations between polar and Cartesian coordinates.

2.1 APPLICATIONS OF THE CORDIC ALGORITHM

2.1.1 Sine and Cosine

The rotational mode CORDIC operation can simultaneously compute the sine and cosine of the input angle. Setting the y component of the input vector to zero reduces the rotation mode result to:

$$\begin{aligned} x_n &= A_n.x_0.\cos z_0 \\ y_n &= A_n.x_0.\sin z_0 \end{aligned} \qquad (2.7)$$

By setting x_0 equal to $1/A_n$, the rotation produces the unscaled sine and cosine of the angle argument, z_0. Very often, the sine and cosine values modulate a magnitude value. Using other techniques (e.g., a look up table) requires a pair of multipliers to obtain the modulation. The CORDIC technique performs the multiply as part of the rotation operation, and therefore eliminates the need for a pair of explicit multipliers. The output of the CORDIC rotator is scaled by the rotator gain. If the gain is not acceptable, a single multiply by the reciprocal of the gain constant placed before the CORDIC rotator will yield unscaled results. It is worth noting that the hardware complexity of the CORDIC rotator is approximately equivalent to that of a single multiplier with the same word size.

2.1.2 Polar to Rectangular Transformation

A logical extension to the sine and cosine computer is a polar to Cartesian coordinate transformer. The transformation from polar to Cartesian space is defined by:

$$\begin{aligned} x &= r\cos\theta \\ y &= r\sin\theta \end{aligned} \qquad (2.8)$$

As pointed out above, the multiplication by the magnitude comes for free using the CORDIC rotator. The transformation is accomplished by selecting the rotation mode with x_0=polar magnitude, z_0=polar phase, and y_0=0. The vector result represents the polar input transformed to Cartesian space. The transform has a gain equal to the rotator gain, which needs to be accounted for somewhere in the system. If the gain is unacceptable, the polar magnitude may be multiplied by the reciprocal of the rotator gain before it is presented to the CORDIC rotator.

2.1.3 General vector rotation

The rotation mode CORDIC rotator is also useful for performing general vector rotations, as are often encountered in motion correction and control systems. For general rotation the 2-dimensionsal input vector is presented to the rotator inputs. The rotator rotates the vector through the desired angle. The output is scaled by the CORDIC rotator gain, which must be accounted for elsewhere in the system. If the scaling is unacceptable, a pair of constant multipliers is required to compensate for the gain. CORDIC rotators may be cascaded in a tree architecture for general rotation in n-dimensions.

2.1.4 Arctangent

The arctangent, $\theta = A\tan(y/x)$, is directly computed using the vectoring mode CORDIC rotator if the angle accumulator is initialized with zero. The argument must be provided as a ratio expressed as a vector (x, y). Presenting the argument as a ratio has the advantage of being able to represent infinity (by setting x=0). Since the arctangent result is taken from the angle accumulator, the CORDIC rotator growth does not affect the result.

$$Z_n = Z_0 + \tan^{-1}(y_0 / x_0)$$

2.1.5 Vector magnitude

The vectoring mode CORDIC rotator produces the magnitude of the input vector as a byproduct of computing the arctangent. After the vectoring mode rotation, the vector is aligned with the x-axis. The magnitude of the vector is therefore the same as the x component of the rotated vector. This result is apparent in the result equations for the vector mode rotator:

$$x_n = A_n \sqrt{x_0^2 + y_0^2}$$

The magnitude result is scaled by the processor gain, which needs to be accounted for elsewhere in the system.

2.1.6 Inverse CORDIC functions

If a CORDIC style computer can generate a function, its inverse can also be computed. Unless the inverse is calculable by changing the mode of the rotator, its computation normally involves comparing the output to a target value.

2.1.7 Arcsine and Arccosine

The arcsine can be computed by starting with a unit vector on the positive x-axis, then rotating it so that its y component is equal to the input argument. The arcsine is then the angle subtended to cause the y component of the rotated vector to match the argument. The decision function in this case is the result of a comparison between the input value and the y component of the rotated vector at each iteration:

$$x_{i+1} = x_i - y_i.d_i.2^{-i}$$
$$y_{i+1} = y_i + x_i.d_i.2^{-i}$$
$$z_{i+1} = z_i - d_i.\tan^{-1}(2^{-i})$$

Where d_i =+1 if $y_i < c$, -1 otherwise.

c is input argument.

Rotation produces the following result:

$$x_n = \sqrt{(A_n.x_0)^2 - c^2}$$
$$y_n = c$$
$$z_n = z_0 + \arcsin\left(\frac{c}{A_n.x_0}\right)$$
$$A_n = \prod_n \sqrt{1 + 2^{-2i}}$$

The arcsine functions as stated above returns correct angles for inputs $-1 < c / A_n x_0 < 1,$ although the accuracy suffers as the input approaches ± 1 (the error increases rapidly for inputs larger than about 0.98). This loss of accuracy is due to the gain of the rotator. For angles near the y-axis, the rotator gain causes the rotated vector to be shorter than the reference (input), so the decisions are made improperly.

The arccosine computation is similar, except the difference between the x component and the input is used as decision function. Without modification, the arccosine algorithm works only for inputs less than $1/A_n$. The arccosine could also be computed by using the arcsine function and subtracting $\pi/2$ from the result, followed by an angular reduction if the result is in the fourth quadrant.

2.1.8 Cartesian to Polar transformation

The Cartesian to Polar transformation consists of finding the magnitude $(r = sqrt(x^2 + y^2))$ and phase angle $(\phi = a\tan[y/x])$ of the input vector, (x, y). These both functions are provided simultaneously by the vectoring mode CORDIC rotator. The magnitude of the result will be scaled by the CORDIC rotator gain, and should be accounted for elsewhere in the system. If the gain is unacceptable, it can be corrected by multiplying the resulting magnitude by the reciprocal of the gain constant.

2.2 EXTENSION TO LINEAR FUNCTIONS:

A simple modification to the CORDIC equation permits the computation of linear functions:

$$\begin{aligned} x_{i+1} &= x_i - 0.y_i.d_i.2^{-i} = x_i \\ y_{i+1} &= y_i + x_i.d_i.2^{-i} \\ z_{i+1} &= z_i - d_i.(2^{-i}) \end{aligned} \qquad (2.8)$$

For rotation mode (d_i = -1 if $z_i < 0$, +1 otherwise) the linear rotation produces:

$$\begin{aligned} x_n &= x_0 \\ y_n &= y_0 + x_0.z_0 \\ z_n &= 0 \end{aligned} \qquad (2.9)$$

This operation is similar to the shift-add implementation of a multiplier, and as multipliers go is not an optimal solution. The multiplication is handy in applications where a CORDIC structure is already available. The vectoring mode (d_i = +1 if $y_i < 0$, -1 otherwise) is more interesting, as it provides a method for evaluating ratios:

$$\begin{aligned} x_n &= x_0 \\ y_n &= 0 \\ z_n &= z_0 + y_0 / x_0 \end{aligned} \qquad (2.10)$$

The rotations in the linear coordinate system have a unity gain, so no scaling corrections are required.

2.3 EXTENSION TO HYPERBOLIC FUNCTIONS

The close relationship between the trigonometric and hyperbolic functions suggests the same architecture can be used to compute the hyperbolic functions. The CORDIC equations for hyperbolic rotations are derived using the same manipulations as those used to derive the rotation in the circular coordinate system. For rotation mode these are:

$$\begin{aligned} x_{i+1} &= x_i + y_i.d_i.2^{-i} \\ y_{i+1} &= y_i + x_i.d_i.2^{-i} \\ z_{i+1} &= z_i - d_i.\tanh^{-1}(2^{-i}) \end{aligned} \qquad (2.11)$$

Where d_i = -1 if $z_i < 0$, +1 otherwise.

Then :

$$\begin{aligned} x_n &= A_n[x_0 \cosh z_0 + y_0 \sinh z_0) \\ y_n &= A_n[y_0 \cosh z_0 + x_0 \sinh z_0) \\ zn &= 0 \\ A_n &= \prod_n \sqrt{1-2^{-2i}} \approx 0.80 \end{aligned} \qquad (2.12)$$

In vectoring mode (d_i = +1 if $y_i < 0$, -1 otherwise) the rotation produces:

$$\begin{aligned} x_n &= A_n \sqrt{x_0^{\,2} - y_0^{\,2}} \\ y_n &= 0 \\ z_n &= z_0 + \tanh^{-1}(y_0 / x_0) \\ A_n &= \prod_n \sqrt{1-2^{-2i}} \end{aligned} \qquad (2.13)$$

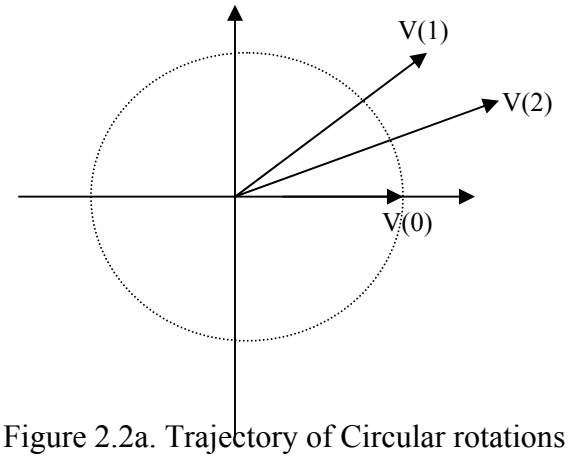

Figure 2.2a. Trajectory of Circular rotations

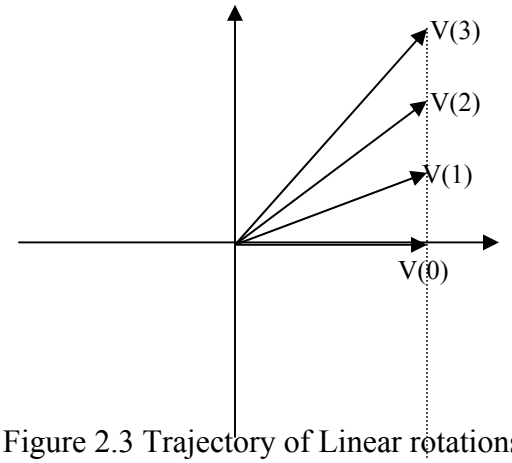

Figure 2.3 Trajectory of Linear rotations

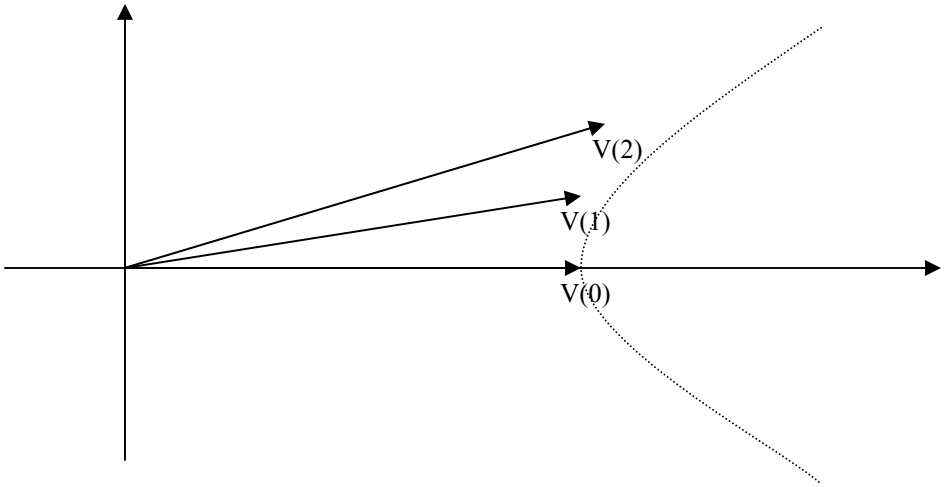

Figure 2.4 Trajectory of Hyperbolic rotations

The elemental rotations in the hyperbolic coordinate system do not converge. However, it can be shown that convergence is achieved if certain iterations (I=4, 13, 40... k, 3k+1,..) are repeated. The hyperbolic equivalents of all the functions discussed for the circular coordinate system can be computed in a similar fashion. Additionally, the following functions can be derived from the CORDIC functions:

$$\tan\alpha = \sin\alpha / \cos\alpha$$
$$\tanh\alpha = \sinh\alpha / \cosh\alpha$$
$$\exp\alpha = \sinh\alpha + \cosh\alpha$$
$$\ln\alpha = 2\tanh^{-1}[y/x] \quad \text{where } x=\alpha+1 \text{ and } y=\alpha-1$$
$$(\alpha)^{1/2} = (x^2 - y^2)^{1/2} \quad \text{Where } x=\alpha+1/4 \text{ and } y=\alpha-1/4$$

It is worth noting the similarities between the CORDIC equations for circular, linear, and hyperbolic systems. The selection of coordinate system can be made by introducing a mode variable that takes on values 1, 0, or -1 for circular, linear and hyperbolic systems respectively. The unified CORDIC iteration equations are then:

$$\begin{aligned} x_{i+1} &= x_i - m.y_i.d_i.2^{-i} \\ y_{i+1} &= y_i + x_i.d_i.2^{-i} \\ z_{i+1} &= z_i - d_i.e_i \end{aligned} \qquad (2.14)$$

Where e_i is the elementary angle of rotation for iteration *i* in the selected coordinate system. Specifically, $e_i = \tan^{-1}(2^{-1})$ for m=1, $e_i = 2^{-i}$ for m=0, and $e_i = \tanh^{-1}(2^{-i})$ for m=-1.This unification, due to Walther, permits the design of a general purpose CORDIC processor.

Chapter 3

SINGULAR VALUE DECOMPOSITION

3. SINGULAR VALUE DECOMPOSITION

The SVD is one of the most important matrix factorizations in linear algebra and the SVD of a matrix $A \in R^{m \times n}$ is a factorization of the form:

$$A = UDV^T \qquad (3.1)$$

Where $U \in R^{m \times m}$ and $V \in R^{n \times n}$ are orthogonal ($U^T U = I$ and $V^T V = I$) and $D \in R^{m \times n}$ is diagonal with nonnegative diagonal elements σ_i. The numbers σ_i are the 'singular values' of A. $U = \{u_1, ..., u_m\}$ are the left singular vectors and $V = \{v_1, ..., v_m\}$ are the right singular vectors.

There are many numerically stable algorithms for computing the SVD such as the Jacobi algorithm, the QR method and the one sided Hestenes method**.** For parallel implementations the Jacobi method is far superior in terms of simplicity, regularity, and local communications. In linear algebra the singular value decomposition (SVD) is an important factorization of a rectangular real or complex matrix, with several applications in signal processing.

Singular values, singular vectors, and their relation to the SVD

A non negative real number σ is a singular value for A if and only if there exist unit-length vectors u in k^m and v in k^n such that

$Av = \sigma u$ And $A^* u = \sigma v$.

The vectors u and v are called left singular and right singular vectors for σ respectively.

A singular value for which we can find two left (or right) singular vectors that are not linearly dependent is called degenerate. Non-degenerate singular values always have unique left and right singular vectors. Consequently, if all singular values of A are non-degenerate and non zero, then its singular value decomposition is unique. Degenerate singular values, by definition, have non-unique singular vectors. Further more if u_1 and u_2 are the two left-singular vectors which both corresponds to the singular value σ, then any linear combination of the two vectors is also a left singular vector corresponding to

singular value σ. The similar statement is true for right singular vectors. Consequently, if A has degenerate singular values, then its singular value decomposition is not unique.

3.1 THE JACOBI SVD ALGORITHM:

Here equation **(**3.1**)** can be rewritten in the following form:

$$A = UDV^T \Leftrightarrow U^T AV = D \qquad (3.2)$$

The Jacobi method exploits to generate the matrices U and V by performing a sequence of orthogonal two sided plan rotations (2D rotations) to the input matrix:

$$\boldsymbol{J}_i^{\theta T} A_i \boldsymbol{J}_i^{\phi} = A_{i+1} \qquad (3.3)$$

With the property that each new matrix A_i is 'more diagonal' than its predecessor. After n iterations, the input matrix $\boldsymbol{A}$ is transformed into the diagonal matrix A_n:

$$A_n = \boldsymbol{J}_n^{\theta T} \boldsymbol{J}_{n-1}^{\theta\ T} \ldots\ldots \boldsymbol{J}_1^{\theta T} \boldsymbol{J}_0^{\theta T} A \boldsymbol{J}_0^{\phi} \boldsymbol{J}_1^{\phi} \ldots\ldots \boldsymbol{J}_{n-1}^{\phi} \boldsymbol{J}_n^{\phi} \qquad (3.4)$$

Which leads to:

$$\begin{aligned} D &= A_n; \\ U &= \prod \boldsymbol{J}_i^{\theta} \\ V &= \prod \boldsymbol{J}_i^{\phi} \end{aligned} \qquad (3.5)$$

The matrices $\boldsymbol{J}_i^{\theta}$ and $\boldsymbol{J}_i^{\phi}$ are 'Jacobi rotation' matrices $J(p,q,\theta)$ of the form:

$$J_{ij} = \begin{cases} J_{pp} = \cos(\theta) \\ J_{qp} = -\sin(\theta) \\ J_{pq} = \sin(\theta) \\ J_{qq} = \cos(\theta) \\ \delta_{ij} elsewhere \end{cases} \rightarrow \begin{bmatrix} 1 \cdots 0 \cdots 0 \cdots 0 \\ 0 \cdots c \cdots s \cdots 0 \\ 0 \cdots -s \cdots c \cdots 0 \\ 0 \cdots 0 \cdots 0 \cdots 1 \end{bmatrix} \qquad (3.6)$$

(Where p < q, $c = \cos(\theta)$ and $s = \sin(\theta)$)

The angles θ and ϕ are chosen in order to solve a 2x 2 SVD problem:

$$\begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta\cos\theta \end{bmatrix} \begin{bmatrix} a_{pp} & a_{pq} \\ a_{qp} & a_{qq} \end{bmatrix} \begin{bmatrix} \cos\phi & \sin\phi \\ -\sin\phi\cos\phi \end{bmatrix} = \begin{bmatrix} a'_{pp} & 0 \\ 0 & a'_{qq} \end{bmatrix} \quad (3.7)$$

By repeating this for all possible pairs (p, q)**,** ***A*** can be effectively digonalized:

Algorithm:

For s=1 ... , S

For p=1, ...,N

For q=p + 1 .. N

Begin

Determine θ and ϕ

A= $J(p,q,\theta)^T A J(p,q,\phi)$

End

The Jacobi algorithm performs 2×2 SVD for all possible pairs (p, q) which is called a sweep, and then repeats that for as many sweeps as necessary for the convergence. The multiplication by J_i^{θ} affects only the two columns p and q, and the multiplication by J_i^{ϕ} affects only the two rows p and q as shown in figure.

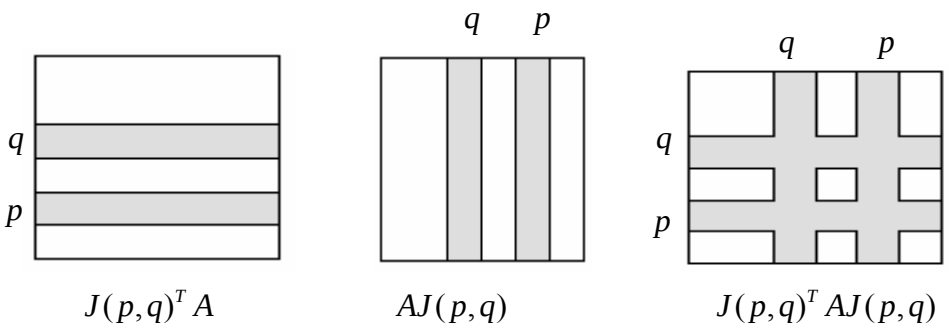

Figure 3.1. Premultiplication $J(p,q)^T A$ effects rows p and q, and post multiplication $AJ(p,q)$ effects columns p and q.

Therefore if M is the matrix size and if M is even then M/2 sub-problems (2 x 2 SVD) can be processed in parallel. To illustrate this suppose M=4 and group the six possible sub problems into three sets:

Set l: $\{(1,2),(3,4)\}$

Set 2: $\{(1,3),(2,4)\}$

Set 3. $\{(1,4),(2,3)\}$

All (p, q) pairs within each set are non-conflicting. Sub problem (1,2) and (3,4) can be calculated out in parallel, likewise sub-problem (1,3) and (2,4) can be executed in parallel as can sub-problem (1,4) and (2,3)**.** This way of ordering sub-problems is called "parallel ordering" it allows the execution of N/2 sub-problems in parallel. The number of sets is N-1 and a sweep consists of N-1 steps of N/2 rotations executed in parallel.

3.2 CORDIC ALGORITHM FOR SVD

For SVD, a $M \times M$ matrix is divided into $\lfloor M/2 \rfloor \times \lfloor M/2 \rfloor$ blocks. Each block is a 2×2 matrix, which can be mapped to a CORDIC processor. The basic operation for Kogbetliantz's method is to apply the two-sided rotation to each 2×2 matrix to nullify the two off-diagonal elements i.e.:

$$\begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta\cos\theta \end{pmatrix}^T \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} \cos\phi & \sin\phi \\ -\sin\phi\cos\phi \end{pmatrix} = \begin{pmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{pmatrix} \qquad (3.8)$$

The values θ and ϕ are obtained using the following equations:

$$\begin{cases} \phi + \theta = \tan^{-1} \dfrac{c+b}{d-a} \\ \phi - \theta = \tan^{-1} \dfrac{c-b}{d+a} \end{cases} \qquad (3.9)$$

The systolic array architecture suitable for implementing Kogbetliantz's approach is shown in Figure 3.3. In this each value, θ and ϕ, is calculated in the diagonal PEs according to equation (3.9), and these are then passed through each off-diagonal PE. Here θ propagates along the rows and ϕ propagates along the columns. The required two-sided rotations are then computed in all PEs in accordance with equation (3.8). After every block has completed this two-side rotation, each pair of values in the both the (2*i*) *th* and (2*i* +1) *th* column is interchanged. The same applies to the rows. A sweep is defined as and occurs when each off-diagonal element is eliminated once. This interchange means that all off-diagonal elements are eliminated before the beginning of the next sweep. Usually, an SVD computation is finished after a pre-defined number of sweeps. In the case of $M \times M$ matrix (M even), this corresponds to $M(M-1)/2$ two sided rotations and it is shown in figure 3.2 for a 4 x 4 matrix.

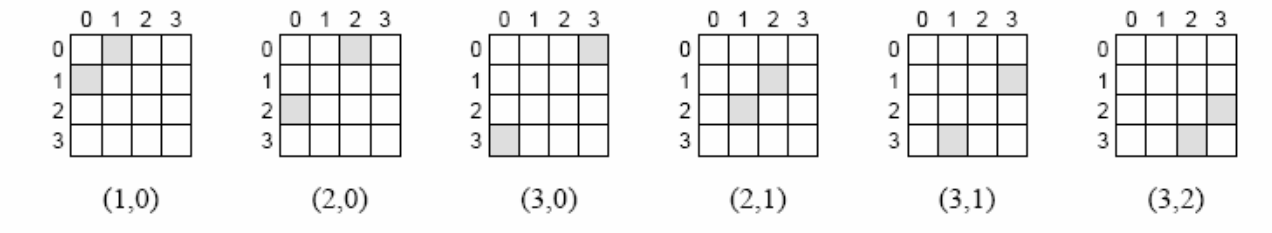

Figure 3.2: Sweep of Jacobi transformations for a matrix of order 4x4

It should be noted that in the case of left-rotation two columns can be calculated in parallel, whilst in the case of right-rotation two rows can be calculated in parallel.

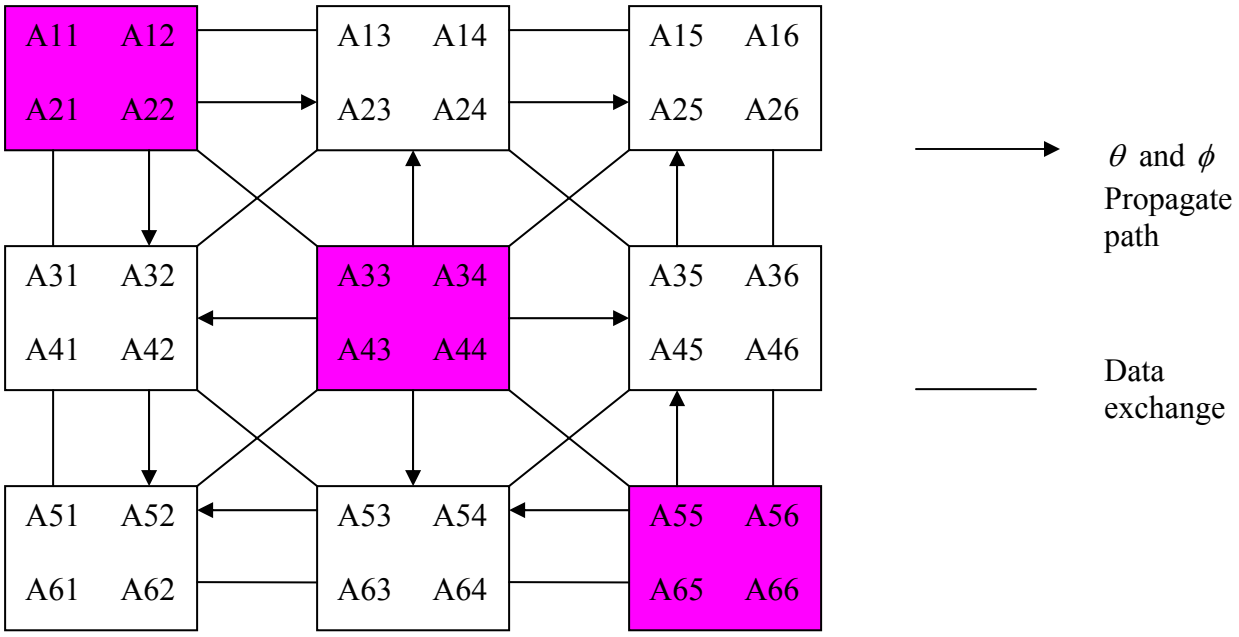

Figure 3.3. Array architecture for kogbetliantz's method

3.3. CORDIC PROCESSOR FOR SVD COMPUTATION

The basic idea behind the CORDIC algorithm is that a rotation is decomposed into a sequence of micro-rotations over the angles α_i, where usually α_i is chosen according to $\tan\alpha_i = 2^{-i}$. The corresponding micro-rotation iterative equations are as follows:

$$\begin{cases} x_{i+1} = x_i - d_i.y_i.2^{-i} \\ y_{i+1} = y_i + d_i.x_i.2^{-i} \\ z_{i+1} = z_i - d_i.\tan^{-1}(2^{-i}) \end{cases} \qquad (3.10)$$

Where $d_i = \pm 1$.

The inverse tangent function is a primitive operation in CORDIC algorithms. Therefore, θ and ϕ in (3.9) can be solved explicitly in (3.10) by setting $d_i = sign(x_i y_i)$. Moreover, within the CORDIC rotation mode equation (3.10) can be used to calculate the matrix vector multiplication shown in equation (3.11) apart from the constant factor scaling i.e.

$$\begin{pmatrix} \tilde{x} \\ \tilde{y} \end{pmatrix} = \begin{pmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha\cos\alpha \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \qquad (3.11)$$

If $d_i = sign(z_i)$ and x_n and y_n are the final outputs from equation (3.10), then the desired outputs $\tilde{x}$ and $\tilde{y}$ in (3.11) are given by:

$$\begin{cases} \tilde{x} = k_n . x_n \\ \tilde{y} = k_n . x_n \end{cases}$$

$$\text{Where} \quad k_n = \prod_{i=0}^{N-1} \cos\ \alpha_i = \prod_{i=0}^{N-1} \frac{1}{\sqrt{1 + 2^{-2i}}} \qquad (3.12)$$

and N is the word length.

In the case of the hybrid architecture x and y are floating-point values and z is a fixed-point number. Let $x_i = m_x . 2^{e_x}$ and $y_i = m_y . 2^{e_y}$, where m_x, m_y, e_x, e_y are the mantissas and exponents of x_i and y_i respectively, with both the mantissa and exponents represented as 2's complement number's. We also assume the inputs x_i and y_i are normalized. Equations (3.10) can then be expressed as equation (3.13).

$$\left\{ x_{i+1} = m'_x . 2^{e'_x} = m_x . 2^{e_x} - d_i . m_y . 2^{e_y} . 2^{-i} = \begin{cases} (m_x - d_i . m_y . 2^{-(e_x + i - e_y)}) 2^{e_x} \; if e_x \geq e_y - 1 \\ (m_x . 2^{-(ey - ex - i)} - d_i . m_y) 2^{e_y - i} \; others \end{cases} \right.$$

$$\left\{ y_{i+1} = m'_y . 2^{e'_y} = m_y . 2^{e_y} + d_i . m_x . 2^{e_x} . 2^{-i} = \begin{cases} (m_y + d_i . m_x . 2^{-(e_y + i - e_x)}) 2^{e_y} \; if ey \geq ex - i \\ (m_x . 2^{-(e_x - e_y - i)} + d_i . m_x) 2^{e_x - i} \; others \end{cases} \right. \qquad (3.13)$$

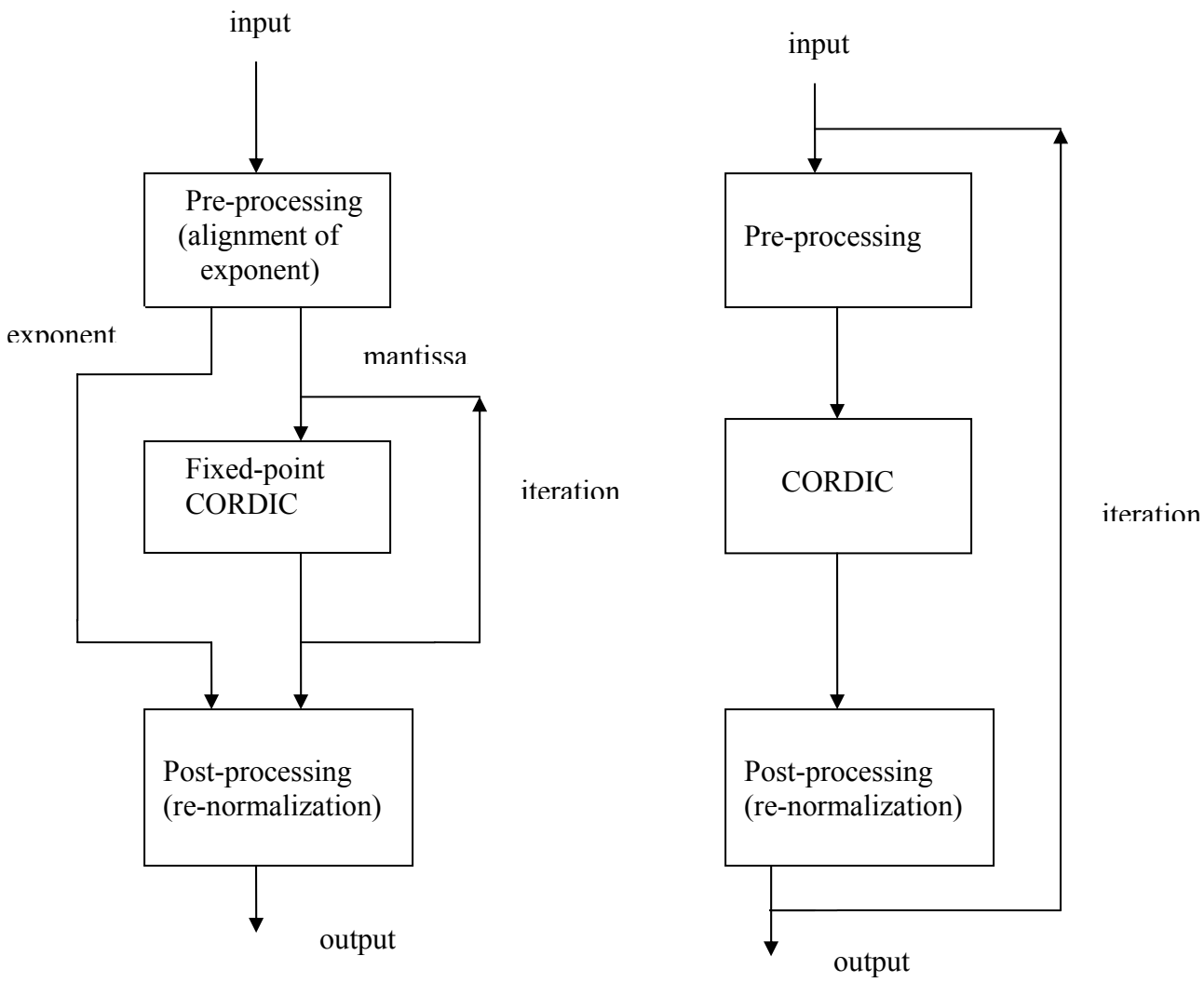

Fig 3.4.Floating-point CORDIC

(a) Global normalization (left) (b) Local normalization

Two options are then available:

- Global floating-point normalization. Here the CORDIC algorithm remains fixed point. In this case it is only at the inputs and outputs of the algorithm that floating-to-fixed and fixed-to-floating point conversions need take place. In this case, the mantissa range must be calculated in advance so that overflow can be avoided. This is shown in Figure 3.4 (a).

- Local floating-point normalization. Here each rotation performs a floating-point shift-add/subtract operation and both $m'_x 2^{e'_x}$ and $m'_y 2^{e'_y}$ need to be re-normalized for the next iteration. This is shown in Figure 3.4 (b).

The advantage of the second approach is the regularity of data scheduling. However, at first sight, its performance appears limited because it requires equal weight for the exponents of both operands in each iteration. Because of this global floating-point normalization has tended to have been used to date. However for SVD computation, the parallel properties of the algorithm can be beneficially exploited to achieve high performance. Architecture of CORDIC module for x- component is shown in figure 3.5.

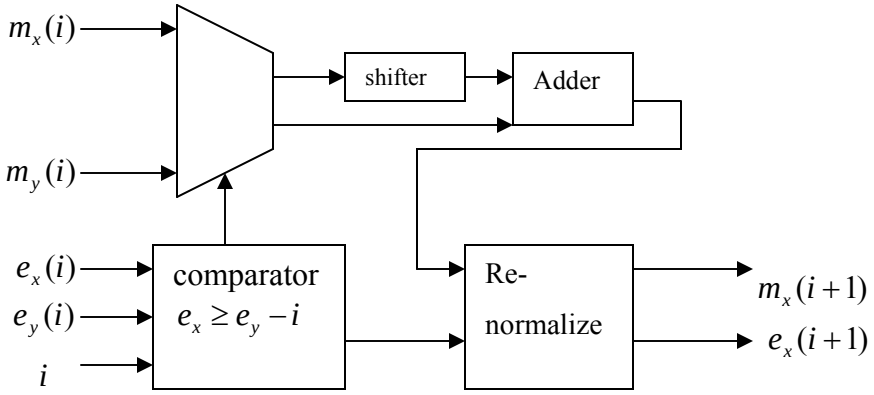

Figure 3.5. Architecture of the x-circuit of CORDIC module

Programmable processor functions

General programmable processor for implementing the SVD PE function can be defined. Whose general operations are summarized as follows:

1) Solve $c \pm b$ and $d \pm a$.

2) Solve $\phi \pm \theta$ using the CORDIC algorithm and then solve θ and ϕ.

3) Solve the left- and right-multiplication using the CORDIC algorithm.

4) Exchange data with neighboring processors.

5) If the pre-defined sweeps have been finished, perform the scale factor correction or else go to step 1.

3.5 MATRIX APPROXIMATION USING SVD:

If the image, when considered as a matrix, has low rank, or can be approximated sufficiently well by a matrix of low rank, then SVD can be used to find this approximation, and further this low rank approximation can be represented much more compactly than the original image. More specifically, suppose we are given an image A which is an $N \times N$ real matrix. Then we first factor it into its SVD representation

$$A = UDV^T \ ,$$

Where D is a diagonal matrix with entries along the diagonal ordered in the decreasing order, and U and V are orthogonal matrices.

Then a rank r approximation to A is the matrix $A_r = U_r D_r V_r^T$

Where D_r is the top-left $r \times r$ sub matrix of D, U_r consists of the first r columns of U, and V_r^T the first r rows of V^T.

The SVD decomposition is interesting because U_r, D_r, V_r^T provide the best rank r approximation to A in the sense of packing the maximum energy from A. Furthermore, for compression, the decomposition is interesting because unlike A which has N^2 entries, the total number of entries in U_r, D_r, V_r^T are only $2Nr + r$. It often turns out that even with small r, the approximation A_r gets most of the energy of A, and is visually adequate. Hence the attractiveness of the method.

Now the use of the SVD for the approximation of the matrix by lower rank is shown in the figure 3.7

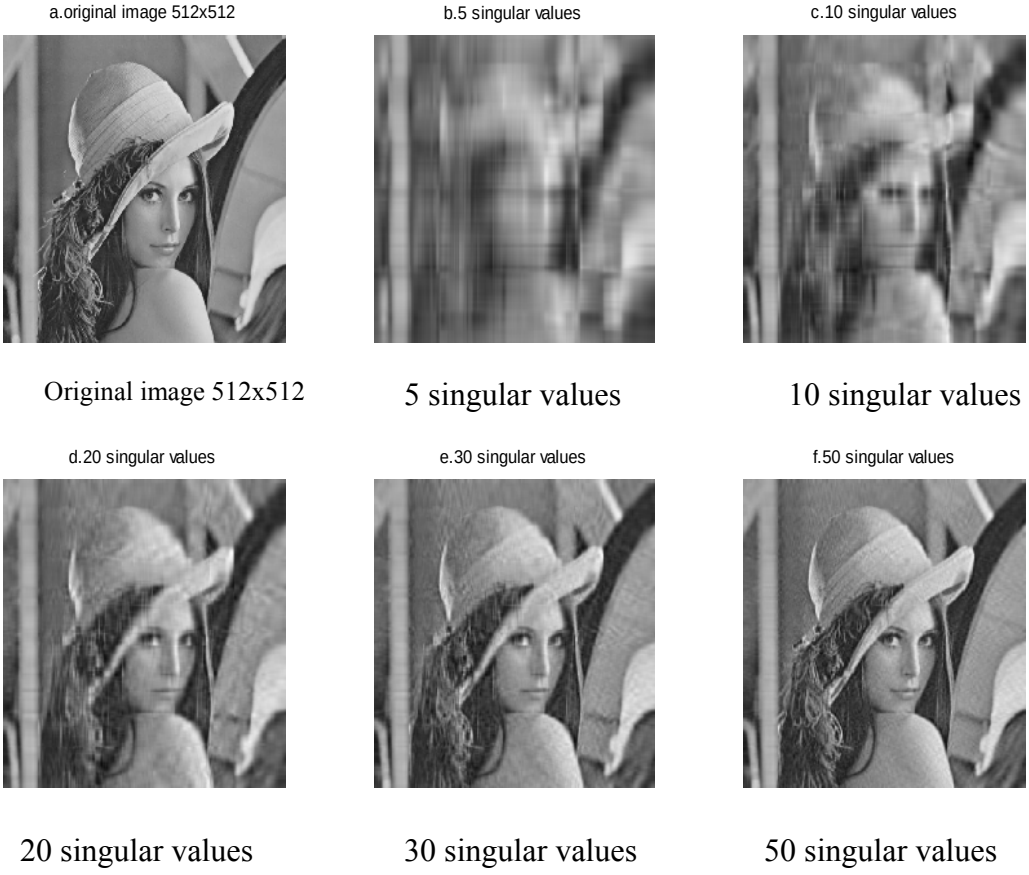

Figure 3.7. Approximation of images using SVD

The first ten iterations actually give the shape of the image, which is a decent approximation. This takes up to 96% less storage space than the original image. First 30 iterations giving a good approximation, we can identify the person with a substantial degree of detail. It requires 90% less storage space than the original image. This is good. Finally, the first fifty iterations give a near perfect image, and yet require 80% less storage space. This is very good.

Chapter 4

DISCTERE COSINE TRANSFORM

4. DISCTERE COSINE TRANSFORM

DCT has become one of the most widely used transform techniques in digital signal processing. The DCT is one of the computationally intensive transforms, which require many multiplications and additions. Many DCT algorithms were proposed in order to achieve high speed DCT. The DCT based on CORDIC [3] algorithm does not need multipliers. Moreover, it has regularity and simple hardware architecture, which makes it easy to be implemented in VLSI. Also, the CORDIC-based DCT algorithm can support the high Performance applications such as HDTV due to its high throughput.

The One-Dimensional DCT

The most common DCT definition of a 1-D sequence of length N is

$$C(u) = \alpha(u)\sum_{x=0}^{N-1} f(x)\cos\frac{(2x+1)u\pi}{2N} \qquad (4.1)$$

for u = 0,1,2,…, N -1. Similarly, the inverse transformation is defined as

$$F(x) = \sum_{u=0}^{N-1}\alpha(u)C(u)\cos\frac{(2x+1)u\pi}{2N} \qquad (4.2)$$

for x =0,1,2,…, N -1. In both equations 4.1 and 4.2 $\alpha(u)$ is defined as

$$\alpha(u) = \begin{cases} \sqrt{\frac{1}{N}} & \text{For } u = 0 \\ \sqrt{\frac{2}{N}} & \text{For } u \neq 0 \end{cases} \qquad (4.3)$$

It is clear from (4.1) that for u = 0 $C(u=0) = \sqrt{\frac{1}{N}}\sum_{x=0}^{N-1} f(x)$. Thus, the first transform coefficient is the average value of the sample sequence. In literature, this value

is referred to as the DC coefficient. All other transform coefficients are called the AC coefficients.

To fix ideas, ignore the $f(x)$ and $\alpha(u)$ component in (4.1). The plot of $\sum_{x=0}^{N-1} \cos \frac{(2x+1)ux}{2N}$ for N =8 and varying values of u is shown in Figure 4.1.

The first the top-left waveform (u =0) renders a constant (DC) value, whereas, all other waveforms (u =1, 2,.. 7) give waveforms at progressively increasing frequencies. These waveforms are called the *cosine basis function*. Note that these basis functions are orthogonal. Hence, multiplication of any waveform in Figure 4.1 with another waveform followed by a summation over all sample points yields a zero (scalar) value, whereas multiplication of any waveform in Figure 4.1 with itself followed by a summation yields a constant (scalar) value. Orthogonal waveforms are independent, that is, none of the basis functions can be represented as a combination of other basis functions.

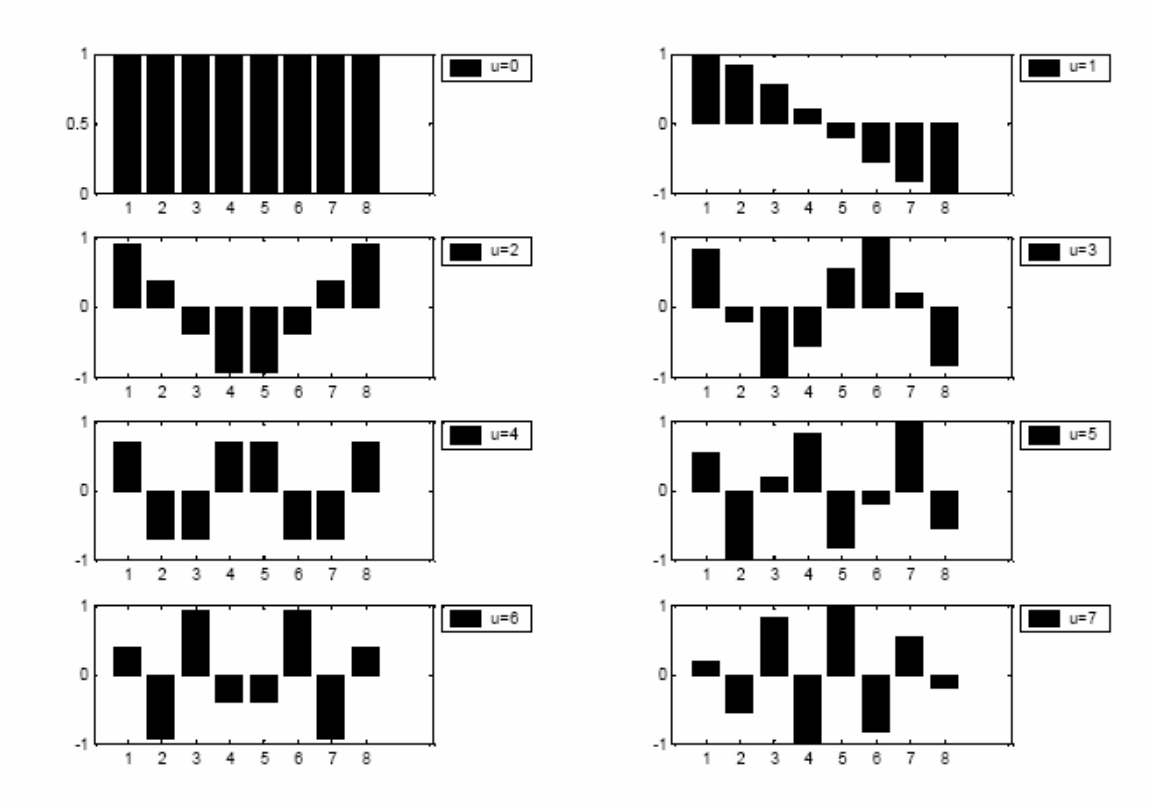

Figure 4.1. One dimensional cosine basis function (*N*=8).

If the input sequence has more than N sample points then it can be divided into sub-sequences of length N and DCT can be applied to these chunks independently. Here, a very important point to note is that in each such computation the values of the basis function points will not change. Only the values of $f(x)$ will change in each sub-sequence. This is a very important property, since it shows that the basis functions can be pre-computed offline and then multiplied with the sub-sequences. This reduces the number of mathematical operations (i.e., multiplications and additions) thereby rendering computation efficiency.

The Two-Dimensional DCT

This section necessitates the extension of ideas presented in the last section to a two-dimensional space. The 2-D DCT is a direct extension of the 1-D case and is given by

$$C(u,v) = \alpha(u)\alpha(v)\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y)\cos\left[\frac{(2x+1)u\pi}{2N}\right]\cos\left[\frac{(2y+1)v\pi}{2N}\right] \qquad (4.4)$$

for u,v =0,1,2,…, N -1 , $\alpha(u)$ and $\alpha(v)$ are as defined in (4.3).

$C(u, v)$ can be computed in two steps by successive 1-D operations on rows and columns. This idea is graphically illustrated in Figure 4.2. The arguments presented can be identically applied for the inverse DCT computation 4.5.

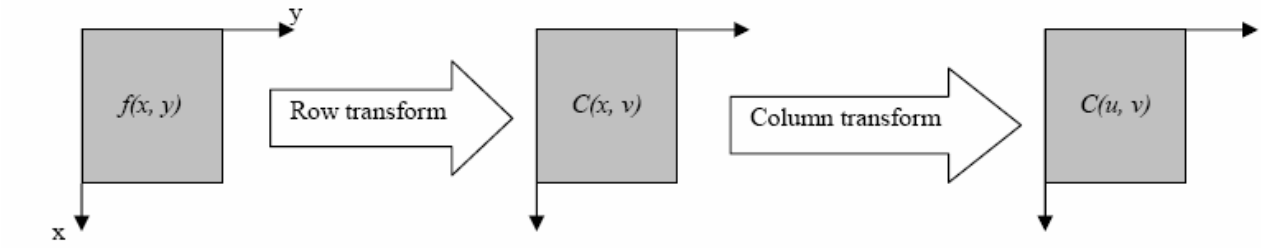

Figure 4.2. Computation of 2-D DCT using separability property.

The inverse transform is defined as

$$f(x,y) = \sum_{u=0}^{N-1}\sum_{v=0}^{N-1}\alpha(u)\alpha(v)C(u,v)\cos\left[\frac{(2x+1)u\pi}{2N}\right]\cos\left[\frac{(2y+1)v\pi}{2N}\right] \qquad (4.5)$$

4.1 CORDIC-BASED DCT ALGORITHM

The two-dimensional DCT for 8×8 sub-matrix is defined as

$$C(u,v) = \frac{1}{4}\alpha(u)\alpha(v)\sum_{x=0}^{7}\sum_{y=0}^{7} f(x,y)\cos\frac{(2x+1)u\pi}{16}\cos\frac{(2y+1)v\pi}{16} \qquad (4.6)$$

$$\alpha(u) = \begin{cases} \frac{1}{\sqrt{2}} & \text{If } u \neq 0 \\ 1 & \text{If } u = 0 \end{cases} \quad \text{And} \quad \alpha(v) = \begin{cases} \frac{1}{\sqrt{2}} & \text{if } v \neq 0 \\ 1 & \text{if } v = 0 \end{cases}$$

Where $f(x,y)$ is the input matrix. Since the two-dimensional DCT is a separable transform, it can be executed by one-dimensional DCT in serial manner. The one-dimensional DCT is expressed as follow.

$$C(u) = \frac{1}{2}\alpha(u)\sum_{x=0}^{7} f(x)\cos\frac{(2x+1)u\pi}{16} \qquad (4.7)$$

To implement the CORDIC based architectures, it is required to decompose the computations in terms of CORDIC operations. $C(0)$ and $C(4)$ can be expressed into (4.8) and (4.9), respectively.

$$C(0) = \{f(0)+f(7)+f(3)+f(4)\}\cos\frac{\pi}{4} + \{f(1)+f(6)+f(2)+f(5)\}\sin\frac{\pi}{4}$$
$$C(4) = \{f(0)+f(7)+f(3)+f(4)\}\sin\frac{\pi}{4} - \{f(1)+f(6)+f(2)+f(5)\}\cos\frac{\pi}{4} \qquad (4.8)\ \&\ (4.9)$$

Where (4.8) and (4.9) are the rotation mode of CORDIC arithmetic. Therefore, in order to compute both $C(0)$ and $C(4)$ we need one CORDIC processor. Similarly, $C(2)$ and $C(4)$ can be obtained by using the rotation mode of CORDIC. For $C(1)$ and $C(7)$, $C(5)$ and $C(3)$, we need four CORDIC processors. Consequently, we can use six CORDIC processors for the 2D-DCT by applying the 1D-DCT two times.

Figure 4.3 shows the 8×1 DCT flow using the six CORDIC processors.

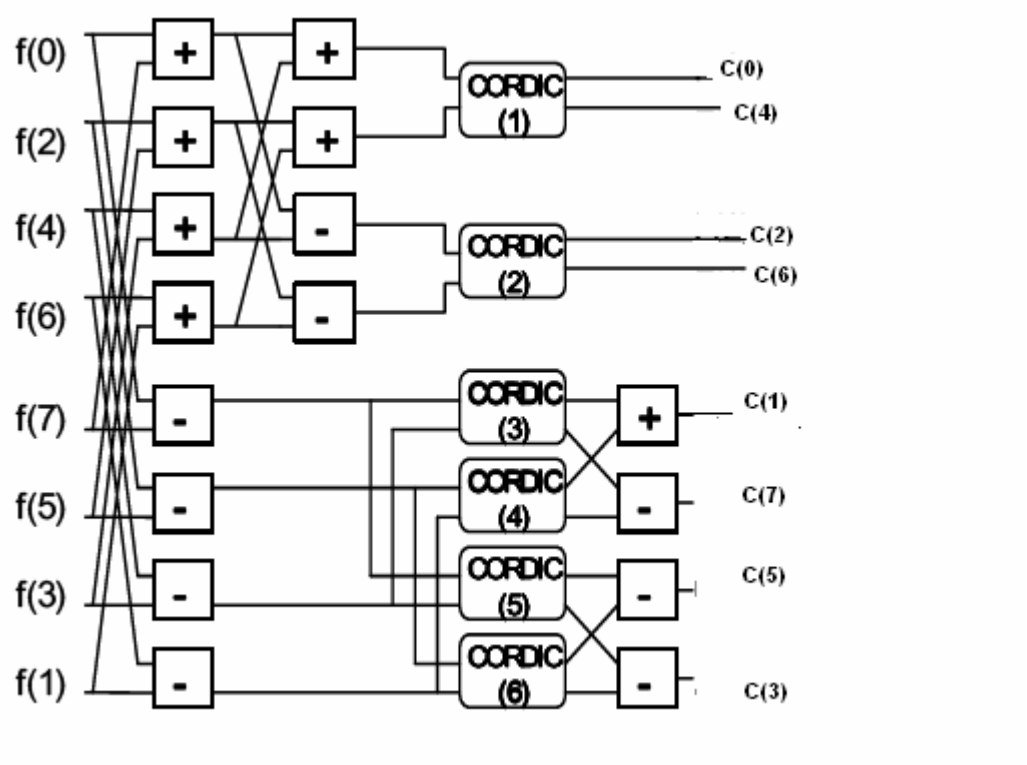

Figure 4.3. 8×1 DCT flow.

Where as rotating angles for CORDIC (1) is $\pi/4$, CORDIC (2) is $3\pi/8$, CORDIC (3) and CORDIC (6) are $7\pi/16$, CORDIC (4) and CORDIC (5) are $3\pi/16$.

Discrete Cosine Transform (DCT) is effectively used to compress a wide variety of images by transferring data into frequency domain. DCT helps separate the image into parts or sub-bands of differing importance with respect to the image's visual quality. For most images, much of signal energy lies at low frequency. Compression is usually achieved by discarding high frequency information since their loss is not easily detected by human visual system.

Now the use of the DCT is shown in the figure 4.4

Original image

Decompressed image

Figure 4.4. Use of the DCT on the images

Chapter 5

RESULTS AND DISCUSSIONS

RESULTS AND DISCUSSIONS

CORDIC algorithm is the rotation algorithm, which provides the iterative method of performing vector rotations by arbitrary angles using shifting and adding operations. Arbitrary angles of rotation are obtained by performing series of successively smaller elementary rotations. CORDIC rotator normally operated in one of the two modes. The first is called the rotation mode and second is vectoring mode. In rotation mode CORDIC rotator rotates the input vector by a specified angle and the resultant plot is shown in the figure 5.1.

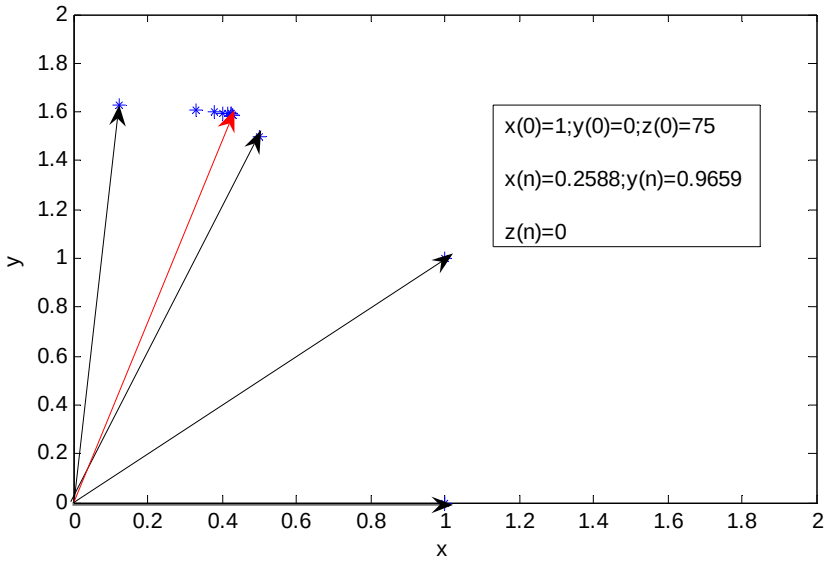

Figure 5.1. CORDIC Rotation mode operations

The rotational mode CORDIC operation can simultaneously compute the sine and cosine of the input angle by setting the y component of the input vector equal to zero. Here in the plot we set the y component of the input vector zero so the y component of the resultant vector gives the $\sin z_0$ and x component gives the $\cos z_0$.

In the vectoring mode, the CORDIC rotator rotates the input vector through whatever angle is necessary to align the resultant vector with the x-axis. The result of the vectoring operation is a rotation angle and the scaled magnitude of the original vector. If the angle accumulator is initialized with zero, it will contain the traversed angle at the end of the iterations.

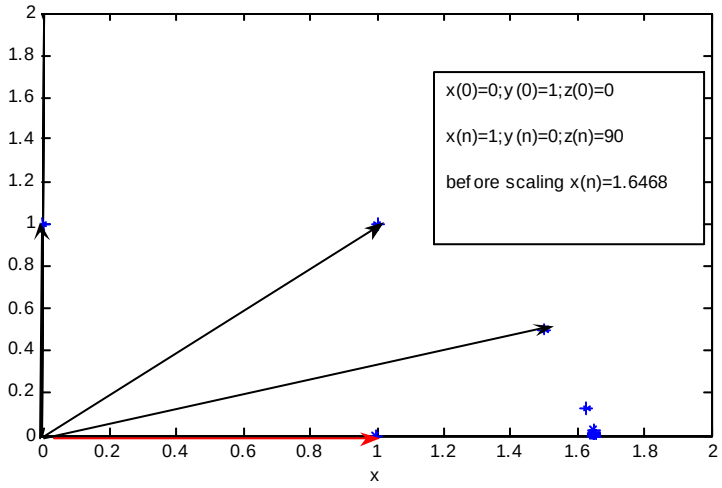

Figure 5.2. CORDIC Vectoring mode operations

Here the input vector is along the y axis so the angle accumulator gives the phase angle 90 degrees and the x component of the resultant vector giving the magnitude unity, of the input vector. This CORDIC algorithm can be used to compute the wide range of functions including trigonometric, linear, logarithmic and exponential function and can also be used to implement the functions such as Singular Value Decomposition and Discrete Cosine Transform.

Singular Value Decomposition is one of the important matrix factorizations in Linear algebra. Jacobi method exploits to compute the SVD by performing a sequence of orthogonal two sided rotations to the input matrix, with the property that each new matrix is more diagonal than its predecessor. CORDIC-based processing element is implemented to perform these two-sided rotations. This processing element contains a CORDIC unit and this is used for both angle solving and for rotation. CORDIC unit used

in vectoring mode for angle solving and used in rotation mode for performing rotation. A sweep is defined as and occurs when each off-diagonal element of the matrix is eliminated once. Usually an SVD computation is finished after a predefined number of sweeps. In the case of $M \times M$ matrix, one sweep corresponds to $M(M-1)/2$ two sided rotations. So $M(M-1)/2$ steps are required to perform one sweep by considering this approach. Whereas the architecture proposed in this thesis, performs one sweep in $M-1$ steps of $M/2$ rotations executed in parallel.

The proposed architecture adopts the parallel ordering method since, in the Jacobi method, one left sided rotation affects only the two columns and right-sided rotation affects only the two rows of the matrix. Therefore if M is the matrix size and even then $M/2$ sub-problems can be processed in parallel. It allows execution of the M/2 sub problems in parallel and hence a sweep consists of M-1 steps of M/2 rotations executed in parallel.

To adopt this parallel ordering method $M \times M$ matrix is divided into $\lfloor M/2 \rfloor \times \lfloor M/2 \rfloor$ blocks. Each block is a 2×2 matrix and mapped to a CORDIC processor. The basic operation is to apply the two-sided rotation to each 2×2 matrix to nullify the two off-diagonal elements i.e.:

$$\begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}^T \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{pmatrix} = \begin{pmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{pmatrix}$$

The values θ and ϕ are obtained using the following equations:

$$\begin{cases} \phi + \theta = \tan^{-1} \dfrac{c+b}{d-a} \\ \phi - \theta = \tan^{-1} \dfrac{c-b}{d+a} \end{cases}$$

In this each value, θ and ϕ, is calculated in the diagonal PEs, and these are then passed through each off-diagonal PE. Here θ propagates along the rows and ϕ propagates along the columns. The required two-sided rotations are then computed in all PEs. After every block has completed this two-side rotation, each pair of values in the both the (2*i*) *th* and (2*i* +1) *th* column is interchanged. The same applies to the rows.

DCT has become the one of the most widely used transform technique in digital signal processing. DCT computation requires many multiplications and additions. Where as the proposed architecture based on CORDIC algorithm does not require any multiplier and it has regularity and simple architecture. This architecture contains six processing elements to compute the DCT for 8x8 matrix. Each processing element consists of one CORDIC unit to perform needed operations. Number of CORDIC units required varies according to size of the matrix. Here each CORDIC unit works only in rotation mode and each CORDIC unit's rotation angle is fixed and the rotating angles for CORDIC (1) is $\pi/4$, CORDIC (2) is $3\pi/8$, CORDIC (3) and CORDIC (6) are $7\pi/16$, CORDIC (4) and CORDIC (5) are $3\pi/16$. 2D-DCT can be computed in two steps by successive 1-D operations on rows and columns.

Hence the CORDIC based processing elements are implemented to compute SVD and DCT. It uses a CORDIC module to perform arithmetic operations and the net result is a flexible computational processing element (PE) for digital signal processing algorithms.

Chapter 6

CONCLUSION

CONCLUSION

By doing this project we are familiar with Coordinate Rotational Digital Computer (CORDIC) algorithm. It is an efficient algorithm suitable to be implemented in DSP algorithms. The aim of the work presented here is to implement CORDIC based processing element for the construction of digital signal processing algorithms. This is a flexible device that can be used in the implementation of functions such as singular-value decomposition (SVD) and Discrete Cosine Transform (DCT). It's calculations for complex arithmetic is simple, elegant. It uses CORDIC module to perform needed arithmetic operations. Besides, since it avoids using multiplication, adopting the CORDIC algorithm can reduce the complexity and the net result is a flexible computational processing element for digital signal processing algorithms.

To implement the CORDIC based architectures for functions like SVD and DCT, it is required to decompose their computations in terms of CORDIC operations. The proposed multiplier free CORDIC architecture for SVD computations requires $M/2 \times M/2$ CORDIC units to process $M \times M$ matrix and the parallel ordering method is adopted to perform $M/2$ computations in parallel. It reduces the number steps required for the computation

The DCT based on CORDIC algorithm does not need multipliers. This proposed architecture for DCT computation requires six CORDIC units to process 8×8 matrix. Moreover, it has regularity and simple architecture. Hence the processing elements for the computation of SVD and DCT are implemented using CORDIC algorithm. This algorithm can also be used for many other DSP algorithms and there is a scope for further improvements in these architectures to increase throughput and hardware sharing by adopting the pipelining method.

REFERENCES

[1] Z.Liu, K.Dickson and J.V. McCanny, “Application-Specific Instruction Set Processor for SoC Implementation of Modern Signal Processing Algorithms,” IEEE Tran on Circuits and Systems-1; Regular papers, vol. 52, no 4, April 2005.

[2]. R.Andraka. “A Survey of CORDIC Algorithms for FPGA Based Computers” – Proc of the 1998 CM/SIGDA Sixth International Symposium on FPGAs, February 22-28, 1998, Monterey, CA, pp.191-200.

[3]. H.Jeong, J.Kim, W.Cho, “Low-Power Multiplierless DCT Architecture Using Image Data Correlation” IEEE Tran on Consumer Electronics, vol 50, no1, Feb 2004

[4]. Y.H. HU, “CORDIC-Based VLSI Architectures for Digital Signal Processing” IEEE Signal Processing magzine, July 1992.

[5]. J. R. Cavallaro and F. T. Luk. "CORDIC Arithmetic for an SVD Processor," Journal of Parallel and Distributed Computing, vol. 5, no. 3, pp. 271-290, June 1988.

[6]. Shen-Fu Hsiao and J. M. Delosme, "Parallel singular value decomposition of complex matrices using multidimensional CORDIC algorithms," IEEE Trans. Signal Processing, vol. 44, pp. 685-697, Mar 1996

[7]. J. R. Cavallaro and F. T. Luk, "Floating-point CORDIC for matrix computations," Proc. IEEE Int. Conf. Computer Design: VLSI in Computers and Processors - ICCD '88 1988, pp.40-42. Washington, DC, USA

[8]. J. Gotze, S. Paul, and M. Sauer, "An efficient Jacobi-like algorithm for parallel eigenvalue computation," IEEE Trans. Comput., vol. 42, pp. 1058-1065, 1993.

[9]. M. D. Ercegovac and T. Lang, "Redundant and on-line CORDIC: application to matrix triangularization and SVD," IEEE Trans. Comput., vol. 39, pp. 725-740, Jun, 1990.

[10]. N. D. Hemkumar and J. R. Cavallaro, "Redundant and on-line CORDIC for unitary transformations," IEEE Trans. Comput., vol. 43, pp. 941-954, Aug, 1994.

[11]. B.G. Lee, “A new algorithm to compute the discrete cosine transform”, IEEE Transactions on Acoustics, Speech and Signal processing, vol. ASSP-32. no.6, pp. 1243-1245, Dec. 1984.